10/544894

REC'D 03 MAY 2004
WIPO PCT

PI 1159475

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

April 28, 2004

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: ***60/490,162***
FILING DATE: ***July 24, 2003***
RELATED PCT APPLICATION NUMBER: *PCT/US04/03609*

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

T. WALLACE
Certifying Officer

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

7/29/2003 KBETEMA1 00000001 60490162
01 FC:2005 80.00 OP

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/69033

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

Express Mail Label No. EL919127935US

00746 U.S. PTO 60/490162 07/24/03

INVENTOR(S)

Given Name (first and middle (if any))	Family Name or Surname	Residence (City and either State or Foreign Country)
Aravind R.	Dasu	Tempe, Arizona
Ali	Akoglu	Tempe, Arizona
Sethuraman	Panchanathan	Gilbert, Arizona

☐ Additional inventors are being named on the _____ separately numbered sheets attached hereto

TITLE OF THE INVENTION (500 characters max)

Algorithm Design for Zone Pattern Matching to Generate Cluster Modules and Control Data Flow Based Task Scheduling of the Modules

CORRESPONDENCE ADDRESS

Direct all correspondence to:

☒ Customer Number 28,529 → Type Customer Number here

Place Customer Number Bar Code Label here

OR

☐ Firm or Individual Name					
Address					
Address					
City		State		ZIP	
Country		Telephone		Fax	

ENCLOSED APPLICATION PARTS (check all that apply)

☒ Specification Number of Pages 36

☐ CD(s), Number

☐ Drawing(s) Number of Sheets

☐ Other (specify)

☐ Application Data Sheet. See 37 CFR 1.76

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT

☒ Applicant claims small entity status. See 37 CFR 1.27.

☒ A check or money order is enclosed to cover the filing fees

☒ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number: 070135

☐ Payment by credit card. Form PTO-2038 is attached.

FILING FEE AMOUNT ($): 80.00

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are: _____

Respectfully submitted,

SIGNATURE _____ Date 7/24/03

TYPED or PRINTED NAME Thomas D. MacBlain REGISTRATION NO. (if appropriate) 24,583

TELEPHONE 602-530-8088 Docket Number: 9138-0106PRV2

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C. 20231.

PTO/SB/17 (01-03)
Approved for use through 04/30/2003. OMB 0651-0032
U.S. Patent and Trademark Office: DEPARTMENT OF COMMERCE

FEE TRANSMITTAL for FY 2003

Effective 01/01/2003. Patent fees are subject to annual revision.

[X] Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT ($) 80.00

Complete If Known	
Application Number	
Filing Date	herewith
First Named Inventor	Dasu
Examiner Name	
Art Unit	
Attorney Docket No.	9138-0106PRV2

METHOD OF PAYMENT *(check all that apply)*

[XX] Check [ ] Credit card [ ] Money Order [ ] Other [ ] None

[XX] Deposit Account:

Deposit Account Number: 070135

Deposit Account Name: Gallagher & Kennedy, P.A.

The Commissioner is authorized to: *(check all that apply)*

[ ] Charge fee(s) indicated below [XX] Credit any overpayments

[XX] Charge any additional fee(s) during the pendency of this application

[ ] Charge fee(s) indicated below, except for the filing fee to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1001	750	2001	375	Utility filing fee	
1002	330	2002	165	Design filing fee	
1003	520	2003	260	Plant filing fee	
1004	750	2004	375	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	80

SUBTOTAL (1) ($) 80

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

	Extra Claims		Fee from below		Fee Paid
Total Claims		-20** =		X	=
Independent Claims		-3** =		X	=
Multiple Dependent					=

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	84	2201	42	Independent claims in excess of 3
1203	280	2203	140	Multiple dependent claim, if not paid
1204	84	2204	42	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($)

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for *ex parte* reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	410	2252	205	Extension for reply within second month	
1253	930	2253	465	Extension for reply within third month	
1254	1,450	2254	725	Extension for reply within fourth month	
1255	1,970	2255	985	Extension for reply within fifth month	
1401	320	2401	160	Notice of Appeal	
1402	320	2402	160	Filing a brief in support of an appeal	
1403	280	2403	140	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,300	2453	650	Petition to revive - unintentional	
1501	1,300	2501	650	Utility issue fee (or reissue)	
1502	470	2502	235	Design issue fee	
1503	630	2503	315	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	
1809	750	2809	375	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	750	2810	375	For each additional invention to be examined (37 CFR 1.129(b))	
1801	750	2801	375	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify) _____

*Reduced by Basic Filing Fee Paid SUBTOTAL (3) ($)

SUBMITTED BY				(Complete *if applicable*)	
Name (Print/Type)	Thomas D. MacBlain	Registration No. (Attorney/Agent)	24,583	Telephone	602-530-8088
Signature				Date	7/24/03

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

If you need assistance in completing the form, call 1-800-PTO-9199 (1-800-786-9199) and select option 2.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Dasu et al.

Filed: Herewith

Title: **Algorithm Design for Zone Pattern Matching to Generate Cluster Modules and Control Data Flow Based Task Scheduling of the Modules**

CERTIFICATE OF MAILING BY EXPRESS MAIL
"Express Mail" mailing label number EL919127935US

Mail Stop Provisional Patent Application
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

I hereby certify that the following correspondence is being deposited in the United States Postal Service as Express Mail on the date shown below in an envelope addressed as shown above.

1. Provisional Application for Patent Cover Sheet (1 sheet);
2. Fee Transmittal for FY 2003 (1 sheet in duplicate);
3. Specification (36 pages plus cover sheet);
4. Check for $80.00; and
5. A return receipt postcard.

1/24/03
Date

Suzanne Shields
Suzanne Shields

GALLAGHER & KENNEDY, P.A.
2575 East Camelback Road
Phoenix, Arizona 85016-9255
Tel. No. (602) 530-8000
Fax. No. (602) 530-8500

Express Mail La[illegible] No. EL919127935US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Provisional Patent Application

Title: Algorithm Design for Zone Pattern Matching to Generate Cluster Modules and Control Data Flow Based Task Scheduling of the Modules

Inventors: Aravind R. Dasu, Tempe, Arizona
Ali Akoglu, Tempe, Arizona
Sethuraman Panchanathan, Gilbert, Arizona

Attorneys for Applicant: Thomas D. MacBlain
Gallagher & Kennedy
2575 East Camelback Road
Phoenix, AZ 85016-9225

Algorithm Design for Zone Pattern Matching to Generate Cluster Modules and Control Data Flow based Task Scheduling of the Modules

Aravind Dasu, Ali Akoglu and Sethuraman Panchanathan

The core underlying principle of a dynamically reconfigurable processor is the optimum reutilization of computing resources. These computing resources can vary from a complex application specific form to a relatively simplistic version for general purpose computing. A necessary form of communication must exist between the processing engines (modules) and the memory unit(s). If the data processing is shared between multiple execution modules, then a secondary form of communication amongst them is necessary. This gives rise to two important issues:

1) Complexity and processing power of the execution modules
2) The burden of coordinating these modules to work as an integral unit.

The more complex the individual modules, the lesser the communication amongst themselves. But this implies decreased flexibility in mapping different algorithmic operations onto a module. If these modules are to be derived based on a given class of applications or algorithms that will be ported onto the processor, then there is an increase in the complexity for identifying such modules. The benefits though are faster execution times.

The second issue deals with the predictability of communication between certain sets of modules with other sets (consumers and producers). The more the predictability, easier is the problem of scheduling the tasks amongst them.

In this report, we will present a technique of resolving both these issues with close to optimal solutions and compare them with existing methods or theories.

Identifying the optimal segments of an algorithm or application for clusterizing.

In [Dasu] we had described a process of narrowing down the search region for identifying clusters. A further analysis showed that a basic block can consist of several zones. This is seen in figure 1, where a small part of the application code of MPEG-4 visual decoding is thoroughly parallelized. What we see is a group of DAGs. The cyclic nature of some graphs have been eliminated by pre-computing the iteration counts (shown in orange circles). It should be noted that exclusion of branch operations in the graphs might result

in a large number of DAGs. The problem of trying to find clusters has been addressed in the field of music analysis and composition. [Emilios] formulates the problem as :

Conceptually we take a similar approach in associating multiple dimensions to a cluster. In order to scientifically determine the amount of parallelization required and the number of resources necessary to perform the set of tasks, we propose that:

1) Identify those zones with the most number of predeterminable iterations, or lying in the most number of loop nestings. Information obtained from running benchmarks can be used in this phase.
2) Populate a table whose row and column header entries are these zones. Along the columns, left to right : most iterative to least iterative. Along the rows, top to bottom : most iterative to least iterative. There can be 3 types of entries in the table. (i) A smiling face indicates a perfect match while migrating from a source to a destination zone or vice versa. (ii) A sad face indicates a partial match (iii) A blank indicates no match.

   The technique of matching such source and destination zones can be performed through tree matching or graph matching. Some researchers have adopted the simpler tree matching method. This is applicable if the source and destination zones are trees. Obviously this severely narrows down the types of applications. For a more generic application agnostic approach, this reduces to a graph matching problem. There have been two notable approaches in this area of research. (i) Graph matching as a graduated assignment problem
   (ii) Graph matching by growing nodes method.

   Graduated Assignment method [Anand]:
   In this approach, a match between 2 graphs is obtained by formulating the differences between the weighted graphs as an objective function. The authors then try to minimize this objective function.

$$E_{wg}(M) = -\frac{1}{2}\sum_{a=1}^{A}\sum_{i=1}^{I}\sum_{b=1}^{A}\sum_{j=1}^{I} M_{ai}M_{bj}C_{aibj}$$

$$\text{subject to } \forall a \sum_{i=1}^{I} M_{ai} \le 1,\ \forall i \sum_{a=1}^{A} M_{ai} \le 1,\ \forall ai\ M_{ai} \in \{0,1\}$$

Here the M_{ai} and M_{bj} are the same matrix (called matching matrix). The C term is the difference in the weights of 2 edges being compared. The summation basically is a combination of every possible edge comparison between the 2 graphs. So, if there is an exact match between 2 edges then the product of the Ms and C will be 1, else it will be 0. Hence the minimum value of the summation represents the maximum number of edge matches between the 2 graphs.

Since a node in a graph can only match up with one node in the other graph, the match matrix should be a permutation matrix.

In classical combinatorial problems, assignment problem corresponds to finding a permutation matrix for a given sample matrix such that the summation of the chosen elements (an element in the sample matrix is chosen if its corresponding entry in the permutation matrix is 1) is maximum. The authors try to convert the given minimization problem into a maximization problem, by expanding the objective function using taylor's series. They then convert the the discrete problem into a continuous version by using a control parameter. This is done by producing an initial match matrix, obtained by exponentiating the error function. This is then subjected to iterative row and column normalization which should result in a doubly stochastic matrix (Sinkhorn's rule). Sinkhorn's rule states that, any positive matrix when iteratively normalized along rows and then along columns will converge into a doubly stochastic matrix. A doubly stochastic matrix is one whose summation along any row or column is positive and less than or equal to one. The newly obtained matrix is again reused in the objective function with a newly increased parameter value.

The complexity of this approach is O(lm) where l and m are the number of edges in the 2 candidate graphs.

corresponding vertices, weighted value of the edge in terms of bit precision and data type. In our problem of graph comparison, the y-axis represents the operations and the x-axis represents the cycle of execution. Hence it is possible for a shifted version of a candidate graph to have a completely different canonical label. Thus completely missing the match. Therefore, flexibility for a match along the time/ clock cycle axis is necessary. We also compare the number of bits required to represent the matrix and the canonical label (MDL encoding) are far less than contemporary methods. These concepts will be illustrated by an example. Consider the following graphs in Figure 1 (1-black with red, 2-black with dotted green and 3-blue).

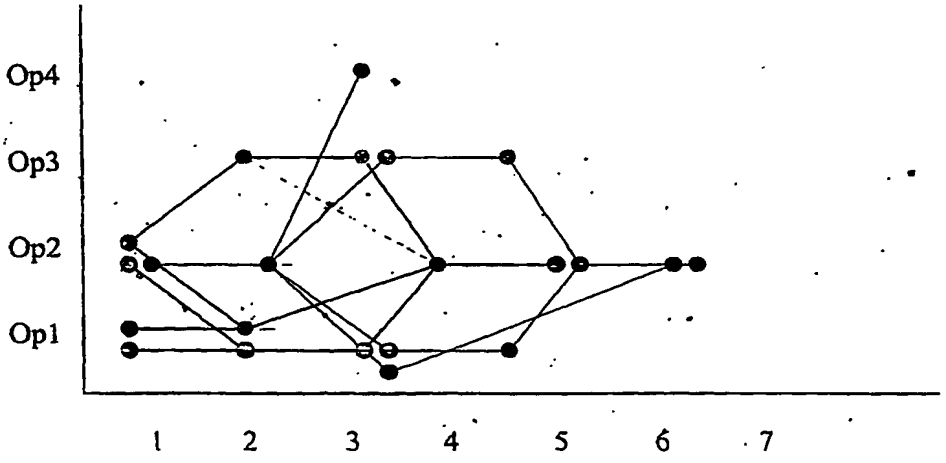


Figure 1 Graphs to illustrate commonality concept

To start with, if there exist multiple nodes at the same (x,y) location, then they are merged into a single node and this information is noted as a part of the weight of a node. If two edges are merged, then that is noted down as a part of weight of the edge, so that the original graph can be reconstructed. A graph is represented in a adjacency matrix whose rows are numbered as (1-1, 1-2, 1-3, 1-4, 2-1, 2-2.....) and so are the columns. The first number (before the hyphen) denotes the cycle and the second number (after the hyphen) denotes the operation. The order in which the non zero elements of the matrix are read out does not affect the

matching process, because the nodes are encoded with respect to fixed x and y axis and we use string comparison techniques. If we use suffix automaton, its possible to detect the largest common sub-graph between graphs 1 and 2. But to obtain the LCS between 1 or 2 and 3, a 'Shift table' is built as follows:

	12-22	22-31	22-33	22-34	31-41	31-62	33-43	41-52	43-52	52-62
11-21	-	-	-	+2,1	-	-	-	-	-	-
12-21	-	+1,1	-	-	-	-	-	-	-	-
12-23	-	-	+1,1	-	-	-	-	-	-	-
21-31	-	-	-	-	+1,1	-	-	-	-	-
21-42	-	-	-	-	-	x	-	x	-	-
23-33	-	-	-	-	-	-	+1,1	-	-	-
31-42	-	-	-	-	-	x	-	+1,1	-	-
33-42	-	-	-	-	-	-	-	-	+2,1	-
42-52	-	-	-	-	-	-	-	-	-	+1,1

The entries on the columns are the edges of graph 3 and the entries along rows are edges of graph 1. The intersections are marked by 2 digits indicate the difference in the start cycles of the edges and the duration of the edge. Only edges with the same source and destination operations and same duration can be compared. The x entries imply that the source and destination operations are the same but the durations are different. Now the entries with same operation combo and duration are counted. This results in Count of +1,1 = 6; Count of +2,1 = 2 and Count of -3,1 = 1. Therefore if graph 1 is shifted by +1 units, there is a guaranteed 6 edge match. This only gives a match number, but the matching itself is performed through the super alphabet string matching technique.

The MDL encoding of the graph can be performed with less than 50% of the number of bits required to encode the same using the method by [UTA]. The

Node growing method:

[3] proposed this method for graph sarches in databases. This involves the formation of a pool of 2 node graphs. The candidates for the nodes in this pool are all possible nodes from the database of graphs to be searched. Each of these small graphs are compared with every candidate graph in the database. The bottom x % of the matches are then pruned. The remaining 2 node graphs are now gown into 3 node graphs. Every possibility is grown. Then the process of comparison is repeated and pruning carried out. The match between 2 graphs is done through comparison of the canonical labels of their adjacency matrices. Various canonification functions can be used to derive the labels. [George] uses the longest possible label. But the drawback of this approach is the lack of weights to the edges. Even without weights complex partitioning schemes are applied to the adjacency matrices to obtain labels for comparison.

Several other universities [Delft], [UCLA] have also adopted this approach without the use of graph comparison through canonical labels.

Both of these approaches suffer from several disadvantages. The graduated assignment method being iterative, gives no indication as to how fast the objective function converges. It is also an approximation approach since it involves maximization of an error term. The growing nodes method (which was also proposed in the qualifying exam) is very slow and cumbersome.

Therefore the following method is proposed, which has the least complexity and hence very fast.

Graph through suffix automaton:

It is well known that suffix automaton is the fastest way O(n) to compare two strings. Prior to approaching the graph comparison problem with suffix automaton, we had approached the comparison through the Largest Common Sub-String algorithm. Although slower, it shares the benefit of finding matches between non-contiguous edges in the graphs. The edges need to be interpreted as alphabets in the string. The edges are represented as a tuple of elements such as

process involves encoding of the edges in the matrix through a system of slope calculations. We start by examining the adjacency matrix. In our example, for every source node we need only 5 bits (3 to indicate the cycle and 2 to identify the position in the cycle). Therefore the number of bits required to represent all source nodes = 7 * 5 = 35. For every source node, there follows pairs of numbers. These are the slope and duration, i.e. Source Node -> Slope 1, Duration1, Slope 2, Duration2,..... The number of such pairs for every source node is an indication of the fan out. So a list of fan out numbers is generated for a graph. In our example that list would be [1,2,3,1,1,1,1]. Therefore the number of bits required to encode this sequence is 2 bits / symbol * 7 symbols = 14 bit. The node sequence would be [11,0,1; 12,-1,1,1,1;21,0,1,1/2,2,2,1; 23,0,1; 31,1,1; 33,-1,1; 42;0,1].

Now the number of bits required to represent the slopes is $\log_2$ (number of unique slopes), and in our example that is = 3. Therefore the number of bits to represent all slopes = 3 * 10 = 30.

The number of bits needed to represent the lengths is $\log_2$ (number of unique lengths), and in our example that is = 1. Therefore the number of bits to represent all slopes = 1 * 10 = 10.

Hence the total number of bits required to represent the entire graph = 35 + 14 + 30 + 10 = 89 bits.

In [UTA] method we would need about 180 bits for the same encoding. Hence our method adheres to the MDL requirement.

Our method of finding the MDL seems quite close to simply encoding the edges as a pair of vertexes (direct encoding). So we explored the option of encoding an edge in terms of its Source Vertex, Slope and Clock Duration. We will explain the process and the overheads involved and then provide an equation, which will indicate any savings in bits compared to direct encoding.

For edges sharing the same Source Vertex, we will represent that group of edges with the Source Vertex followed by a Slope and CD for every fan out Destination Vertex. The corresponding fan out number is also stored. The format therefore is:

FO: SV,Slope1,CD1,Slope2,CD2.....

And the savings in bits =

(# SVertexes)* {$\log_2$(Max FO) + $\log_2$(# Ops + # Clocks)}

+ (# of Edges)* {$\log_2$(Max Slope) + $\log_2$(Max CD)}

- 2*(# of Edges)*$\log_2$(# Ops + # Clocks)

This also needs to be compared to Run Length Coding, which is quite useful in encoding long sequences of numbers with a large number of zeros.

However, clearly the most optimal method depends on the adjacency matrix, which cannot be predicted for all classes of applications.

Although the process of finding the Largest common sub-graph through Suffix Automaton is of low complexity, yet the preparation of such a 'Shift Table' involves all element comparison which if both matrices have equal number of edges (n) then, will be of complexity $O(n^2)$. Moreover, SA can only be used for exact (complete) sub-graph matching. Accommodation of partial matches can only be performed using the Largest Common Subsequence algorithm, which is of complexity order $O(n^2)$. Therefore to reduce complexity of the approach, we modified the population technique of the adjacency matrix. When an element of the matrix is populated, the tag count of row (source) index is incremented. Initially all row index tags are assumed to be 0. The stored column index is subtracted from the current column index and the difference (jump) is stored. The stored column index is now replaced by the current column index. Therefore for every tagged row index, there are 3 pieces of information. (i) tag count (ii) array of jumps (iii) stored column index. This process helps in quickly scanning only the necessary populated (non-zero) elements from the adjacency matrix. We do that by checking for the first row that is tagged. Then using the jump array, we pick up the next element, decrement the tag count and proceed till the tag count is zero. Then check for the next tagged row.

The string of edges (elements) obtained this way will now be sorted using an efficient algorithm such as Merge Sort whose complexity is much lesser than $O(n^2)$. An edge consists of 4 basic elements (Source Clock, Source Operation, Destination Clock, Destination Operation). The sorting criteria will be: same SO-DO combination starting from lowest SO; if SOs are the same and DOs are

different, then lower DO takes priority. If SOs are different, then lower SO takes priority. Then once a sorted list is obtained, a set of m^2 bins are created, where m is the number of operations. The bins are arranged in a queue with the same priority criteria as used for the sorting algorithm. For example, if we had 4 operations (1,2,3 &4), then the queue of bins would be (11,12,13,14,21,22,23,24.....) where 12 will mean: SO =1 and DO =2. Now we need to place the edges into appropriate bins. This is done by comparing an element's (edges) SODO pair with the head of the queue's SODO. If a match occurs, then place the element into that bin. Else if a mismatch occurs, then the head of the queue is shifted to the right by one position and the SODO comparison is again made, and queue head shifted till a match is met. Once all elements are placed in the bins, then a similar process is followed for the other graph to be compared, and a second bin queue is obtained.

To find the largest Common Sub-graph, we only need to operate on corresponding pairs of bins (bins with same SODO) from both queues, where both bins have elements in them. For example a sample bin pair queue is shown in Figure 2.

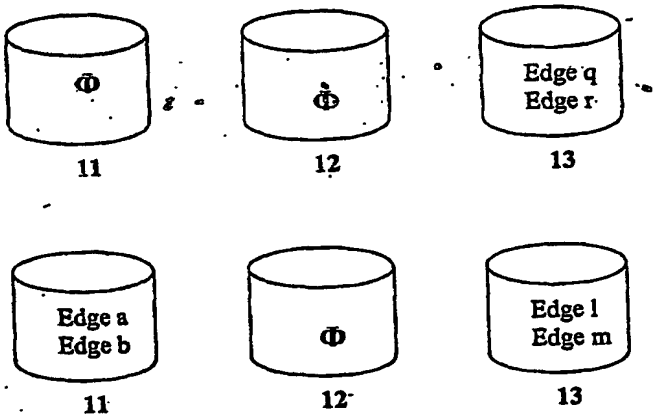


Figure 2 Bin pair queue

We perform a LCS algorithm on the chosen bin pairs (for example on bin pairs 13 and 13). Although LCS has a $O(n^2)$ complexity, yet the number of elements being compared are much lower than the whole set of edges. But care must be taken to arrange the edges in each bin in the order of increasing Source Clock cycles. This will simulate an alphabet sequence such as 'abcaaba...' where the repeated alphabets are shifted versions of the first occurrence.

He resulting set of Largest Common Subsequences from the chosen bin pairs are collectively called the Largest Common Sub-graph.

We confidently believe that for any application, this method is the fastest and least complex approach to finding the largest common sub-graph.

Constraints of reconfiguration distance can be applied at the bin choosing step.

3) A probability table of source to destination jump is built. This helps in pruning out zones which are less likely to occur and have low matching with other zones.
4) The zones are now designed in hardware such that the edit distance between zones portable on a module involves a minimum amount of logic switching.

Task Scheduling

We now address the second issue of task scheduling. In the graph matching problem, we can include branch operations to reduce the number of graphs. This can be done, if one of the paths of a branch operation leads to a very large graph compared to the other path, or is a subset of the other path. This still leaves us with the problem of conditional task scheduling with loops involved. Several researchers have addressed task scheduling and one group has also addressed loop scheduling with conditional tasks. In this report we will not discuss all the relevant work done by other research groups. Instead we focus on some important ones and propose a method most suitable for the purpose of reconfiguration and compare it with the contemporary methods.

[Chekuri's] paper discusses the earliest branch node retirement scheme. This is applicable for trees and s-graphs. An s-graph is a graph where only one path has weighted nodes. In this case, it is a collection of DAGs representing basic blocks which all end in branch nodes, and the options at the branch nodes are: exit form the whole graph or exit to another branch node as shown in Figure 3. The two DAGs are

in blue and violet and the branch nodes are in red with probabilities of exiting the system being p and q. Such a graph is scheduled on a generic set of processing elements such that the branch node with the least schedule length to sum of exit probabilities from the system (upto that branch node) is scheduled as early as possible. The denominator implies how many sub-graphs 'can be completed'. The numerator implies 'how fast this can be done' or 'what is the least amount of time it takes to do it'. Therefore, a low rank indicates that a large number of sub-graphs can be completed in a small amount of time. This is a form of list based scheduling where they try to minimize the expected time of completion.

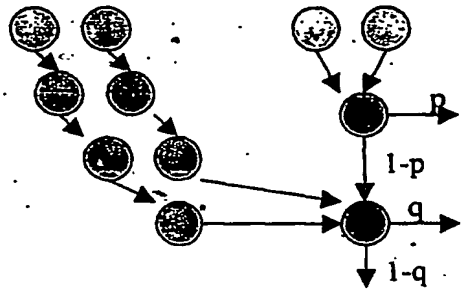


Figure 3 Early retirement schedule

The problem with this approach is that its applicable only to 'small, as defined by the authors' graphs and also restricted to S-graphs and trees. It also does not consider nodes mapped to specific processing elements.

[Jha's] paper addresses scheduling of loops with conditional paths inside them. This is a good approach as it exploits parallelism to a large extent and uses loop unrolling. But the drawback is that the control mechanism for having a knowledge of 'which iteration's data is being processed by which resource' is very complicated. This is useful for one or two levels of loop unrolling. It is quite useful where the processing units can afford to communicate quite often with each other and the scheduler. But in our case, the network occupies about 70% of the chip area [Andre] and hence cannot afford to communicate with each other too often. Moreover the granularity level of operation between processing elements is beyond a basic block level and hence this method is not practical. And within

a processing element, since the reconfiguration distance (edit distance) is more important, fine scale scheduling is compromised because the benefits with the use of very fine grain processing units is lost due to high configuration load time.

[Mooney's] paper discusses a 'path based edge activation' scheme. This basically means, if for a group of nodes (which must be scheduled onto the same processing unit and whose schedules are affected by branch paths occurring at a later stage) we know ahead of time the branch controlling values, then we can at run time prepare all possible optimized list schedules, for every possible set of branch controller values. In the following simple example shown in Figure 4, the nodes in gray need to be scheduled on the same processing unit. The branch controlling variable is b which can take values of 0 or 1. In case it takes a 0, the branch path in red is taken, else the path in green is taken. In the case where we can know at run time, yet ahead of time of occurrence of the branch paths, the value of 'b', we can prepare schedules for the 3 grey nodes and launch either one, the moment b's value is known.

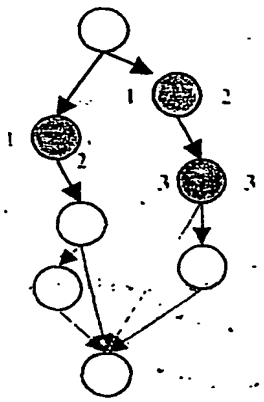


Figure 4 Path based edge activation

This method is very similar to the partial critical path based method proposed by [Pop]. It involves the use of a hardware scheduler and is quite well suited for our application. But we need to add another constraint to the scheduling: the amount of reconfiguration or the edit distance.

[Pop's] paper tackles control task scheduling in 2 ways. The first is partial critical path based scheduling, which is discussed above. Although they do not assume that the value

of the conditional controller is known prior to the evaluation of the branch operation. They also propose the use of a branch and bound technique for finding a schedule for every possible branch outcome. This is quite exhaustive, but it provides an optimal schedule. Once all possible schedules have been obtained, the schedules are merged. The advantages are that it is optimal, but its has the drawback of being quite complex. It also does not consider loop structures.

We propose to utilize a combination of these methods with modifications to suit our needs of reconfiguration. At the coarse level of scheduling, we will exploit all those clusters, whose iteration counts are known at compile time. Loop unrolling is performed to the extent that the edit distance to the next possible configuration on the same module is not disturbed to a large extent. For scheduling clusters on the same module, if the branch controlling conditions are known at run time, but prior to the branch execution, we will employ [Mooney's] method. For all other possibilities we will use the branch and bound technique, but with 2 additional constraints. One is that of reconfiguration distance which will affect the bounding, and the other is for loops whose life times are not known at compile time. All dependent nodes (clusters) are in a wait state till the loop is completed. This indicates a break point in the schedule chart.
Therefore in our approach the factors being considered are:

data dependency

- execution time (can minimize by parallelizing and faster clock)
- reconfiguration time (must be minimized as it is a significant factor)
- communication time (in more likely cases of branch prediction, it should not be a problem, but in the unlikely cases it might be significant)

Task scheduling for control data flow graphs

Problem statement:
Given a control-data flow graph, we need to arrive at an optimal schedule. Sections 1 and 2 introduce the CDFG and the resources. Section 3 discusses the methodology to arrive at the optimal schedule. This includes PCP scheduling and improvisations for specific

situations. It also talks about the technique used for merging of various schedules. Section 4 discusses reconfiguration. Section 5 talks about issues involved with loops.

1. Control-Data Flow Graph:

   A directed cyclic graph has been used to model the entire application. It is a polar graph with both source and sink nodes. The graph can be denoted by G (V, E). V is the list of all processes that need to be scheduled. E is the list of all possible interactions between the processes. The processes can be of three types: Data, communication and reconfiguration. The edges can be of three types: unconditional, conditional and reconfiguration. Here a simple example with no reconfiguration and no loops has been shown in Figure 5. Reconfiguration will be dealt with in section 7. Loops will be handled in section 8.

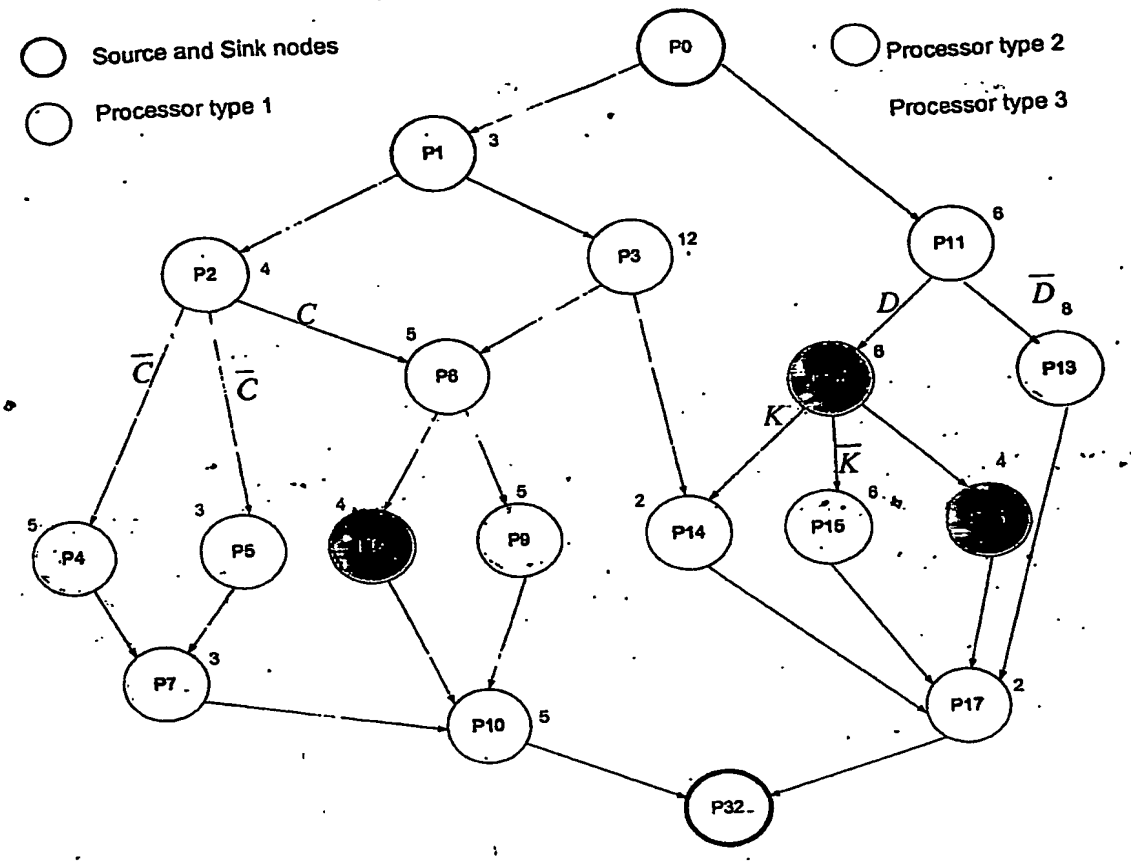

Source and Sink nodes
Processor type 1
Processor type 2
Processor type 3
P0
P1
3
P2
4
P3
12
P11
6
C
$\overline{C}$
$\overline{C}$
5
P8
D
$\overline{D}$
8
P13
6
K
$\overline{K}$
4
5
P4
3
P5
4
P9
2
P14
P15
6
P7
3
P10
5
P17
2
P32

Figure 5 An Example of a Control Data Flow Graph.

In the above graph, each of the circles represents a process. Sufficient resources are assumed for communication purposes. All the processes have an execution time associated with them, which has been shown alongside each circle. If any process is a control-based process, then the various values to which the condition evaluates to are shown on the edges emanating from that process circle.

2. Resources:

Let the number of resources allocated for Process of type PEi be Ni. In Figure 1, the following configuration has been assumed. There are three processors PE1, PE2 and PE3. N1 = N2 = N3 = 1, one for each type of process.

Determination of number of resources for each type of process:

Our approach:

Obtain the sub-graphs for every possible path. For example, the graph in Figure 1 is used to obtain the path for DCK (sub-graph shown in Figure 5-a).

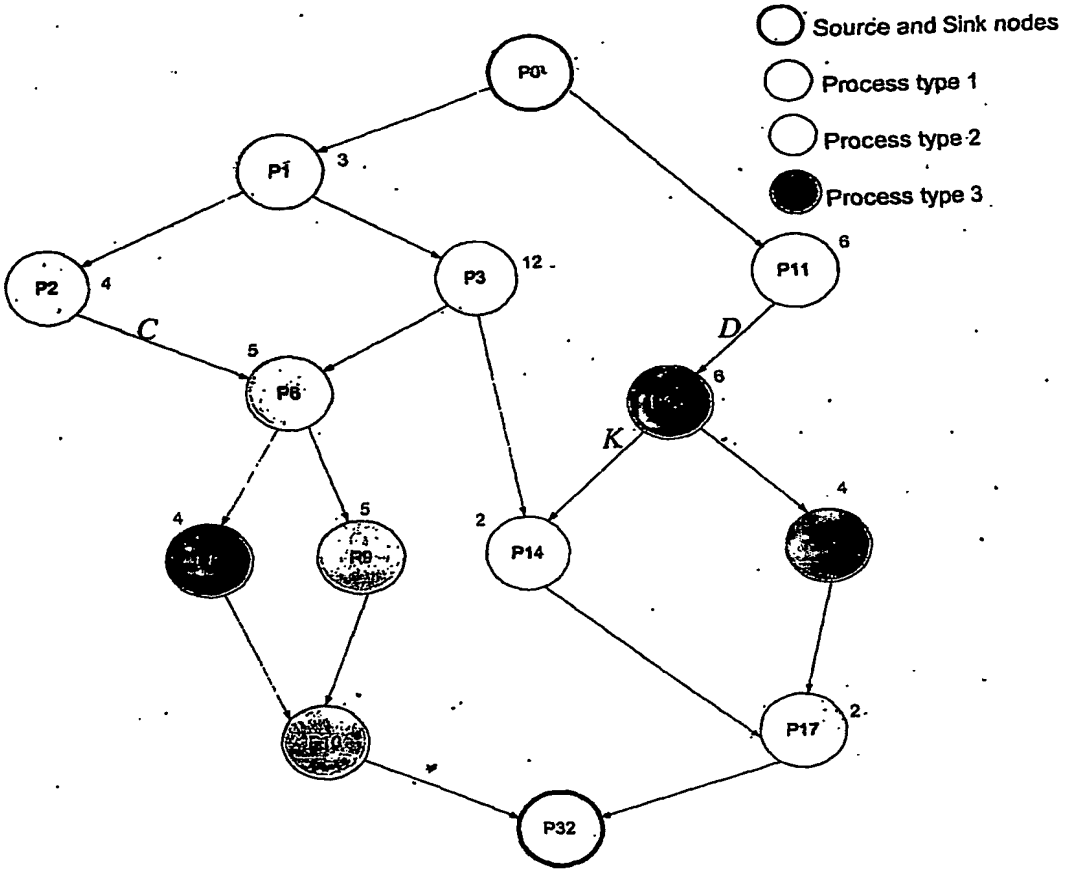


Figure 5-a Sub-graph for condition DCK

For this particular example graph, 6 such sub-graphs are obtained, one for each of the conditions: DCK, $DC\overline{K}$, $D\overline{C}K$, $D\overline{C}\overline{K}$, $\overline{D}C$, $\overline{D}\overline{C}$.

For any sub-graph, we can determine the number of units of each type of processor. This is done by isolating the nodes corresponding to a processor type. For example, in the DCK example taken above, 3 more graphs can be obtained as shown in Figure 6 and 7. It might not make sense to apply this policy to all the 6 sub-graphs. Therefore only those deemed as most likely to be taken should be considered. In case an unlikely path does end up being taken, the clock speed for the general purpose computing resources (the programmable LUTs) must be designed suitably if real-time requirements exist.

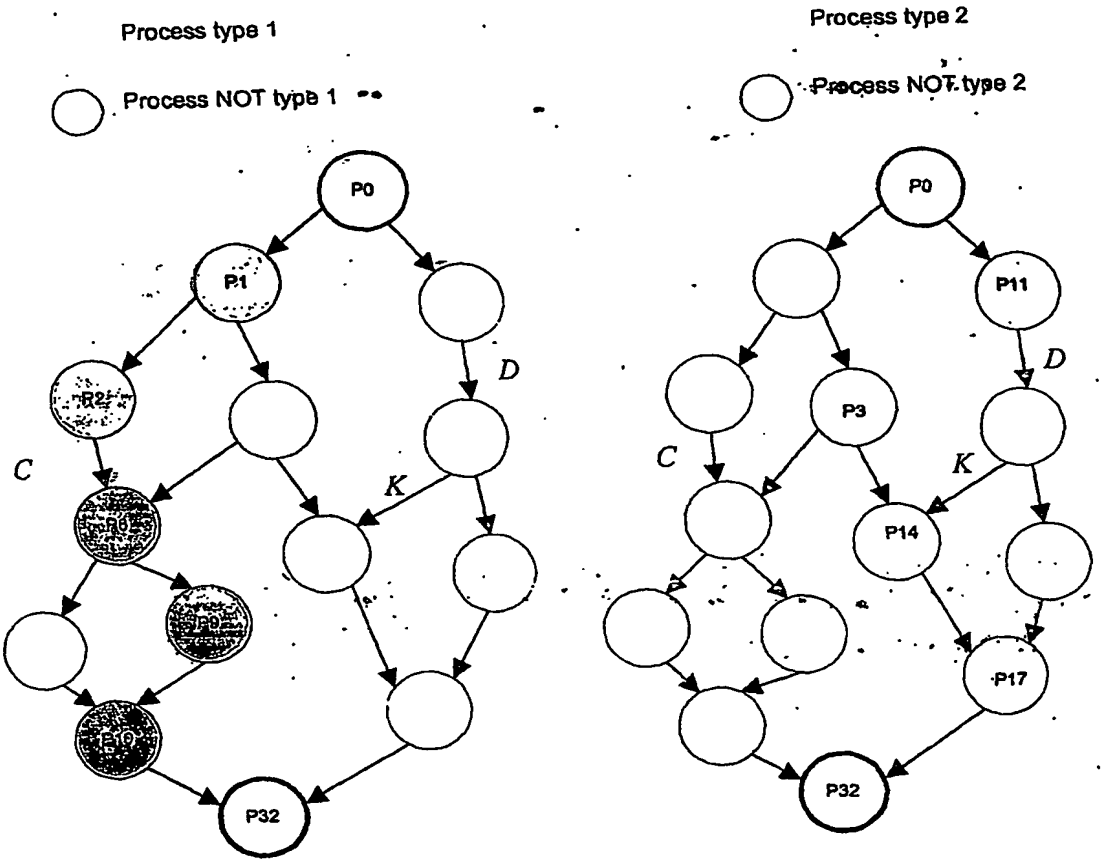


Figure 6 Graphs obtained for individual process types 1 and 2

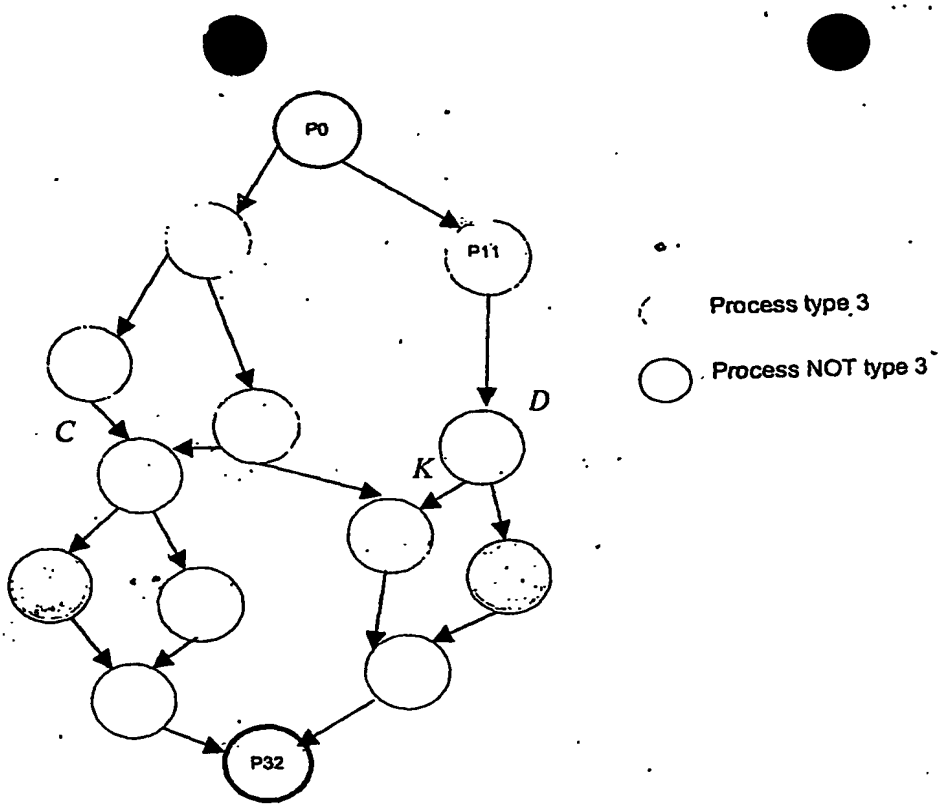


Figure 7 Graphs obtained for individual process type 3

For a graph with only a specific process type highlighted, its possible to obtain the number of processor units, by identifying the critical path and separating the graph into processes in the critical path and those outside it. For example if we had a graph as shown in Figure 8, with the critical path marked in dotted line arrows, we would group processes P1, P5, P8, P9, P10, P12 and P13 into the primary group. And we would place all the other processes into another group called the secondary group. If the combined execution time in the primary group is say Tp and the combined execution time in the secondary group is Ts, then we check for the ratio of Tp : Ts. If the ratios are close to 1:1, then it means that most likely, maximum benefit can be obtained by scheduling each of the groups onto 2 parallel processors. If the ratio is 1:x where $x \geq 1$, then in the secondary group, a critical path is identified. Thus the secondary group is similarly split into 2 groups. We proceed in this divide and conquer method till a 1:1:1... or a close ratio is obtained. But if Tp : Ts is x : 1, then

there would be an underutilization of resources if additional processing units are allocated. In this case we might be better off using a single resource allocation.

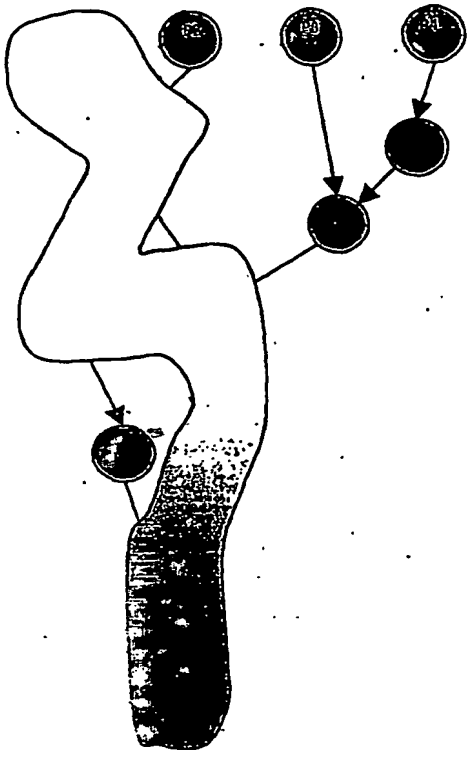


Figure 8 Divide and Conquer method of determining number of parallel units

3. Methodology:
   i. Use PCP scheduling to determine the delays for each possible path of the CDFG and arrange the list of paths in descending order of the delays.

ii. Perform branch and bound based scheduling (which need not be done for every path to reduce the complexity).

iii. Once the final list of all schedules is ready, merge all the schedules by respecting data and resource dependencies.

PCP scheduling:

PCP is a modified list-based scheduling algorithm. The basic concept in a Partial critical path based scheduling algorithm is that if we have a situation as shown in Figure 9 below, where Processes P_A, P_B, P_X, P_Y are all to be mapped onto the same resource say Processor Type 1. P_A and P_B are in the ready list and a decision needs to be taken as to which will be scheduled first. λ_A and λ_B are times of execution for processes in the paths of P_A and P_B respectively, but which are not allocated on the Processors of type 1 and also do not share the same type of resource.

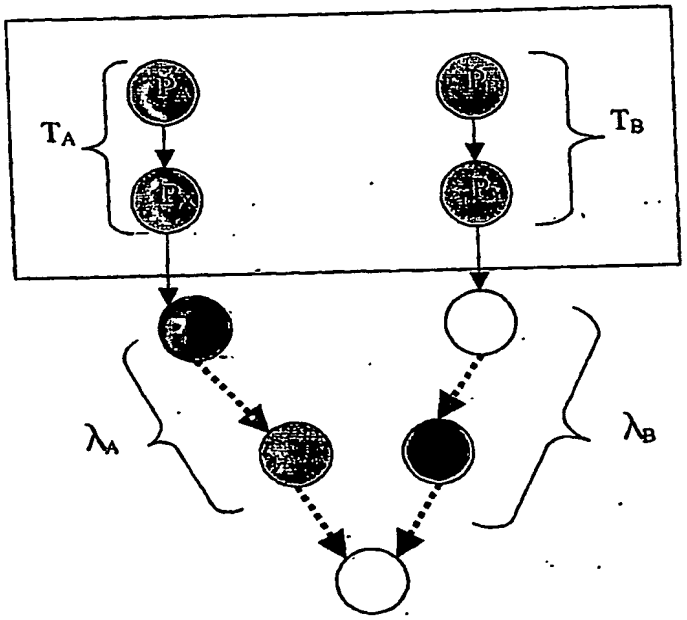


Figure 9 PCP based scheduling

If P_A is assigned first, then the longest time of execution is decided by the

Max $(T_A + \lambda_A, T_A + T_B + \lambda_B)$

If P_B is assigned first, then the longest time of execution is decided by the

Max $(T_B + \lambda_B, T_B + T_A + \lambda_A)$

The best schedule is the minimum of the two quantities. This is called the partial critical path method because it focuses on the path time of the processes beyond those in the ready list. Therefore if λ_A is larger than λ_B, a better schedule is obtained if Process A is scheduled first. But this does not consider the resource sharing possibility between the processes in the path beyond those in the ready list. A simple example (Figure 10) shows that if $T_A = 3$, $T_B = 2$, $\lambda_A = 7$, $\lambda_B = 5$, where in processes in the λ_A and λ_B sections share the same resource, say Processor type 2, then scheduling Process A first gives a time of 15 and scheduling B first gives a time of 14. But both the critical path and PCP as proposed by Pop suggest scheduling B first.

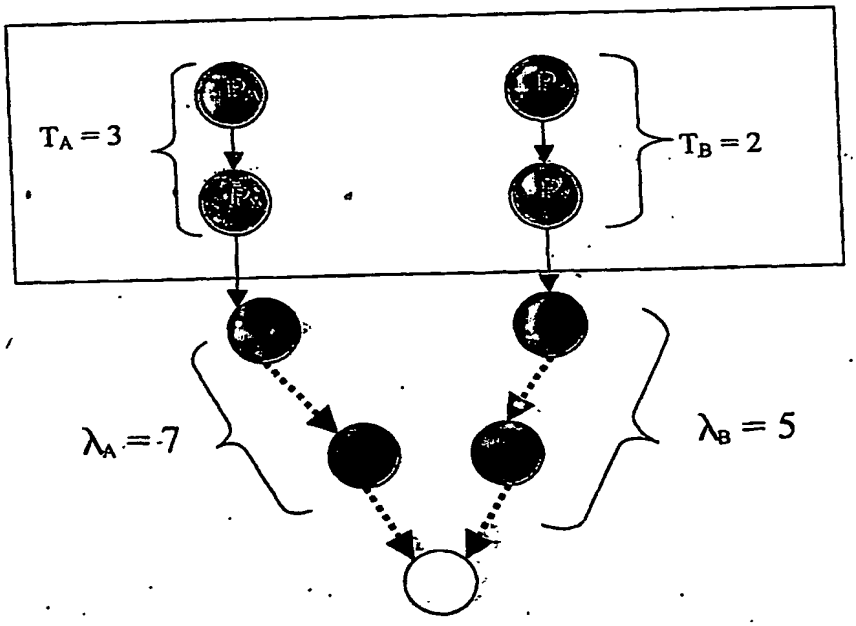


Figure 10 PCP scheduling with Resource dependencies in the Partial path region

The difference is because, if the resource constraint of the post ready list processes is considered, the best schedule is a min of 2 max quantities:

Max (T_B, λ_A) & Max (T_A, λ_B).

Pop [Pop] uses the heuristic obtained from PCP scheduling to bound the schedules in a typical branch and bound algorithm to get to the optimal schedule. But branch and bound algorithm is an exponentially complex algorithm in the worst-case. So there is a need for a lesser complex algorithm that can produce near-optimal schedules. From a higher view point of scheduling we need to limit the need for BB scheduling as much as possible.

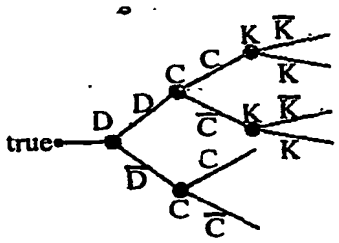
Initially, the control variables in the CDFG are extracted. Let c1, c2,.....,cn be the control variables. Then there will be at most 2^n possible data-flow paths of execution for each combination of these control variables from the given CDFG. An ideal aim is to get the optimal schedule at compile time for each of these paths. Since the control information is not available at compile time, we need to arrive at an optimal solution for each path with every other path in mind. This optimal schedule is arrived at in two stages. First the optimal individual schedule for each path is determined. Then each of these optimal schedules is modified with the help of other schedules.

Stage 1: There are m=2^n possible Data Flow Graphs (DFG's). For each DFG, the PCP scheduling is done. Then, the DFG's are ordered in the decreasing order of their total delays.

An optimal solution can be obtained by doing branch and bound scheduling for each of these PCP scheduled DFG's. But branch and bound is a highly complex algorithm with exponential complexity. In this case, this complex operation needs to be done 2^n times, where n is the number of control variables, which increases the complexity way beyond control. Hence Branch and bound is done only when it is essential to do so. Then BB scheduling is done for DFG1, which has the largest delay. For DFG2, the PCP delay is compared with the BB delay of DFG1.

If the PCP delay is smaller, then the PCP scheduling is taken as the optimal schedule for that path. If not, then the BB scheduling is done to get the optimal schedule. It makes sense to do this, as the final delay of each DFG after modification is going to be close to the delay of the worst delay path. In the same way, the optimal schedule is arrived at for each of the DFG.

Stage 2: Once the optimal schedule is arrived at, a schedule table is initialized with the processes on the rows and the various combinations of control variables on the column. A branching tree is also generated, which shows the various control paths. This contains only the control information of the CDFG. There exists a column in the schedule table corresponding to each path in this branching tree. The branching tree is shown in Figure 11. The path corresponding to the maximum delay is taken and the schedule for that corresponding path is taken as the template (DCK'). Now the DCK path is taken and the schedule is modified according to that of DCK'. This is done for all the paths. The final schedule table obtained will be the table that resides on the processor.



Length of the optimal schedule for the alternative paths through the graph in Fig. 1	
$D \wedge C \wedge \overline{K}$	39
$\overline{D} \wedge C$	39
$D \wedge C \wedge K$	38
$D \wedge \overline{C} \wedge \overline{K}$	32
$D \wedge \overline{C} \wedge K$	31
$\overline{D} \wedge \overline{C}$	31

Figure 11 Branching tree

The pseudo code of this process is summarized in Figure 12.

```
Algorithm:
Task schedule (G(V,E), CTRL_VARS[N], PE = {PE1,PE2.....PEM})
For each combination of CTRL_VARS do
{
    Generate a DFG Gsub(V,E,CTRL_VARS[I]) which is a sub-graph of G(V,E).
    Only the nodes and edges in the control flow corresponding to the current
    combination of CTRL_VARS are included in this sub-graph.
    Generate the PCP schedule of Gi. Let the schedule be PCP_sched[I] and the
    delay be PCP_delay[I].
}
Sort PCP_sched and PCP_delay and Gsub in decreasing order of PCP_delay[I].

Generate the Branch and bound schedule for Gsub[0], the sub-graph with the worst
PCP_delay. Let the schedule be BB_sched[I=0] and the delay be BB_delay[I=0].
Initialize worst_bb_delay = BB_delay[0]

For all the other sub-graphs do
{
    if (PCP_delay[I] < worst_bb_delay) then
            BB_sched[I] = PCP_sched[I];
            BB_delay[I] = PCP_delay[I];
    else
            Generate BB_sched[I] and BB_delay[I];
            If (BB_delay[I] > worst_bb_delay[I]) then
                    Worst_bb_delay = BB_delay[I];
}

Generate the branching tree with the help of the G(V,E). In this branching tree, the
edge represents the choices (K and K') and the node represents the variable (K)
Initialize the current path to the one leading from the top to the leaf in such a way
that the DFG corresponding to this path gives the worst_bb_delay. The path is
nothing but a list of edges tracing from the top node till the leaf.
```

Figure 12 Selective use of PCP and BB algorithms

We also observe that processes with large execution times have a greater impact on the schedule than the shorter processes. Hence, we decided to schedule large processes in a special way. The shorter processes can be scheduled using the PCP scheduling algorithm. Since PCP scheduling is done for most of the processes, the complexity stays closer to O(N), where N is the number of processes to be scheduled.

a) Identify the first set of processes that need to be scheduled onto the same processor which are computationally complex. Let's call them MP1, MP2....(Macro process 1 etc.)
b) Schedule all the processes till these macro processes in the data flow graph using PCP scheduling.
c) Calculate the estimated execution time of the smaller processes to find the start time of each of the macro process.
d) Determine the next set of such macro processes in the DFG. Let's call them MP_sub1, MP_sub2...
e) For processes amidst these two sets of macro processes, PCP scheduling is used.
f) For processes occurring after the second set of macro processes, the execution times are added up to get the total execution time.
g) Now, determine the order of execution of these processes by estimating the worst-case execution time in each case and selecting the best amongst them.
h) After this scheduling, the block after the second set of macro processes is taken as the current DFG and steps a-g re implemented.
i) Step h is repeated till the end of DFG is reached.

Schedule merging algorithm:

In the schedule table there are some columns representing paths that are complete and some that are not. The incomplete paths can be now referred to as parent paths of possible complete paths.

In the example shown in Figure 1, we see that for earliest evaluation of all conditional variables (viz. D, C, K) it is necessary to evaluate D first, then C and then K. Therefore the tree of possible paths is as shown in Figure 11. Now, while creating the schedule table, initially only consider the full possible paths i.e. , the 6 paths listed in Figure 11. Perform scheduling by the suggested algorithm. This will fill these columns. Then create the remaining column of partial paths (i.e., D, D^C, ...etc). These are now just empty columns. Now if a process has the same

start times in multiple columns, then push it into the parent empty column. This approach tries to obtain the worst case delay and merge all paths to that timeline. Since the D^C^K(bar) path had the worst case optimal delay, all other full paths were adjusted to match this path. But it is also necessary to consider the probability of the occurrence of all the full paths (6 of them). Then prune out the bottom 10% of the paths, that is, disregard those full paths whose probability of occurrence is less than a threshold value when compared to the path with most probable occurrence.

Then a path is selected from the remaining ones, whose probability of occurrence is the highest. This will be the new reference to whom all the remaining paths will adjust to. Now it is likely that these chosen full paths and the disregarded full paths, share certain partial paths (parent paths). Therefore, while allocating the start times for the processes that fall under these shared partial paths, we must allocate them based on the worst (most delay consuming) disregarded path which needs (shares) these processes. While performing schedule merging, all data dependencies must be respected.

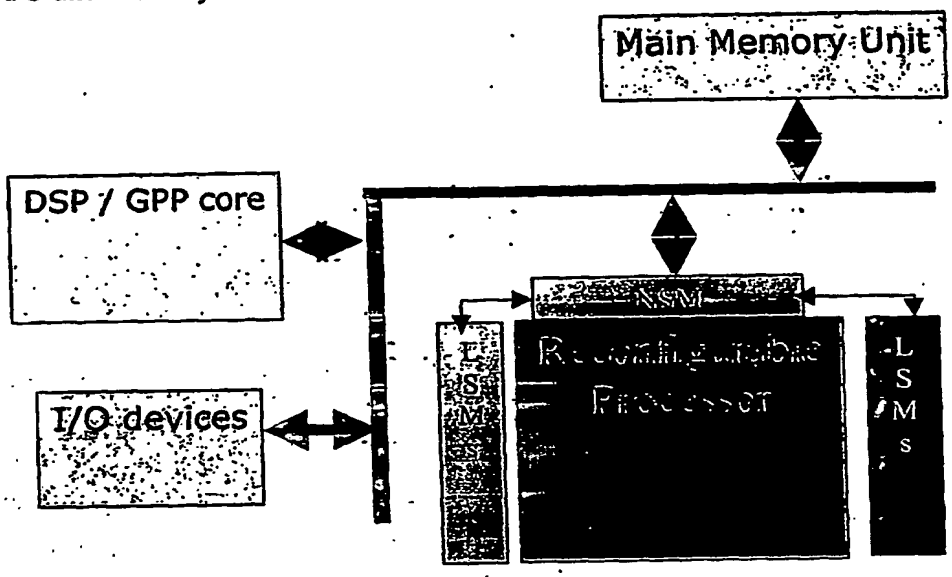
4. Reconfiguration:

   Incorporating Reconfiguration time into Control flow graphs involves the following steps:

   i. Special edges are added onto the control flow graphs. These graphs exist between a similar set of processes, which will be executed on the same processor with or without reconfiguration.
   ii. Reconfiguration times affect the worst-case execution time of loopy codes. So this has to be taken care of, when loopy codes are being scheduled.
   iii. Care needs to be taken to schedule the transfer of reconfiguration bit-stream from the main memory to the processor memory.

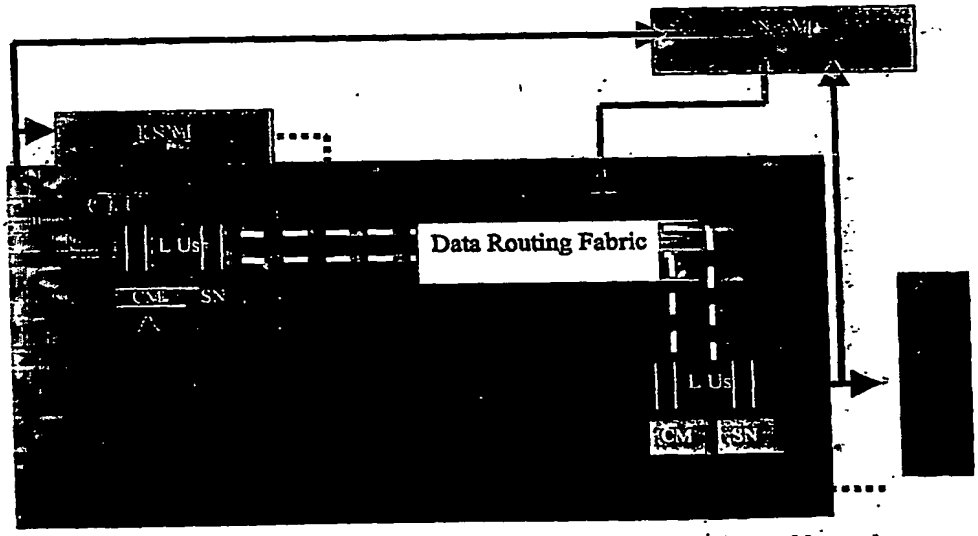
Before the concepts involved in Loop based or influenced scheduling are explained, a brief overview of the architecture of the reconfigurable unit must be presented. In

Figure 13, the interfacing of the Reconfigurable unit with the host processor and other I/O and memory modules is shown.



LSM = Logic Schedule Manager; NSM = Network Schedule

Figure 13 Overview of the system architecture



CLU = Configurable Logic Unit; LU = Logic Units; SN = Switching Network
CM = Configuration Memory; LSM = Logic Schedule Manager

Figure 14 The internals of the Reconfigurable Unit

The Network Schedule Manager (Figure 14) has access to a set of tables, one for each processor. A table consists of possible tentative schedules for processes or tasks that must be mapped onto the corresponding processor subject to evaluation of certain conditional control variables.

The Logic Schedule manager schedules and loads the configurations for the processes that need to be scheduled on the corresponding Processor ie. all processes that come in the same column (a particular condition) in the schedule table.

In PCP scheduling, since the scheduling of the processes in the ready list depends only on the part of the paths following those processes, the execution time of the processes shall initially conveniently include the configuration time.

Once a particular process is scheduled and hence removed from the ready list, another process is chosen to be scheduled based on the pcp criteria again. But this time the execution time of that process is changed or rather reduced by using the reconfiguration time, instead of the configuration time. Essentially, for the first process that is scheduled in a column,

the *completion time = execution time + configuration time*.

For the next or successive processes,

completion time = predecessor's completion time + execution time + reconfiguration time

Assuming that once a configuration has been loaded into the CM, the process of putting in place the configuration is instantaneous, it is always advantageous to load successive configurations into the CM ahead of time. This will mean a useful latency hiding for loading a successive configuration.

The reconfiguration time is dependent on two factors:

1) How much configuration data needs to be loaded into the CM (Application dependent)
2) How many wires are there to carry this info from the LSM to the CM (Architecture dependent)

The Network Schedule Manager should accept control parameters from all LSMs. It should have a set of address decoders because to send the configuration bits to the

Network fabric consisting of a variety of switch boxes, it needs to identify their location. Therefore for every column in the table, the NSM needs to know the route apriori. We must NOT try to find a shortest path at run time. For a given set of processors communicating, there should be a fixed route. If this is not done then, the communication time of the edges n the CDFG cannot be used as constants while scheduling the graph.

For any edge the,

communication time = *a constant and uniform configuration time* + *data transaction time.*

5. Loop-based scheduling:

Case 1: Solitary loops with unknown execution time. Here, the problem is the execution time of the process is known only after it has finished executing in the processor. So static scheduling is not possible.

Solution:

(Assumption) Once a unit generates an output, this data is stored at the consuming / target unit's input buffer.

Consider the following scheduled chart (Figure 15). Each row represents processes scheduled on a unique type of unit (Processor). Let P1 be the loopy process.

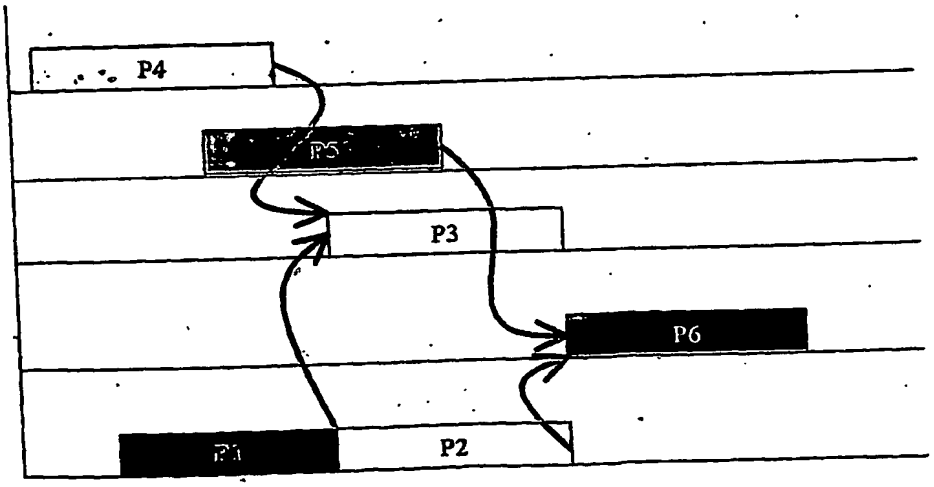


Figure 15 Scheduled process charts with resource and data dependency

From the above figure we see that

P3 depends on P1 and P4,

P2 depends on P1,

P6 depends on P2 and P5.

If P1's lifetime exceeds the assumed lifetime (most probable lifetime), then all dependents of P1 and their dependents (both resource and data) should be notified and the respective NSM and LSM entries delayed. Of course, this implies that while preparing the schedule tables, 2 assumptions are made.

1) The lifetimes of solitary loops with unknown execution times are taken as per the most probable case obtained from prior trace file statistics.
2) All processes that are dependent on such solitary loop processes are scheduled with a small buffer at their start times. This is to provide time for notification through communication channels about any deviation from assumption 1 @ run time.

If assumption 1 goes wrong, the penalty paid is:

Consider the example in Figure 9 where 2 processes in the ready list are being scheduled based on PCP. Now by PCP method if $\lambda_A > \lambda_B$ and P1 & P2 do not share the same resource, then PA is scheduled earlier than PB. WE have assumed that λ_A is due to most probable execution time of Loop P1. But at runtime if Loop P1 executes lesser # of times than predicted and therefore resulting in λ_A being < λ_B, then the schedule of PA earlier than PB results in being a mistake.

We should be able to calculate the time difference between both possible schedules.

We do not at this point propose to repair the schedule because all processes before P1 have already been executed. And trying to fit another schedule at run time, requires intelligence on the communication network which is a burden. But on the brighter side, if @ run time Loop P1 executes more # of times than predicted, then λ_A will still be > λ_B. Therefore the assumed schedule holds true.

Case 2: A combination of two loops with one loop feeding data to the other in an iterative manner.

Solution: Consider PA feeding data to PB in such a manner. For doing static scheduling, if we loop unroll them and treat it in a manner of smaller individual processes, then it is not possible to assume unpredictable number of iterations. Therefore if unpredictable number of iterations is assumed in both loops, then memory foot-print could become a serious issue.

But an exception can be made. If both loops at all times run for the same number of iterations, then the schedule table must initially assume the most probable number of iterations and schedule PA,PB,PA,PB and so on in a particular column. In case the prediction is exceeded or fallen short off, then the NSM and LSMs must do 2 tasks:

1) If the iterations exceed expectations, then all further dependent processes (data and resource) must be notified for postponement and notified for scheduling upon the iterations completion with an appropriate difference in expected and obtained @ run time, schedule times. If the iterations fall short of expectations, then all further schedules must only be preponed.
2) Since the processes PA and PB should denote single iteration in the table, their entries should be continuously incremented @ run time by the NSM and the LSMs. The increment for one process of course happens for a predetermined number of times, triggered off by the schedule or execution of the other process. For example in Figure 16, we see that PA = 10 cycles, PB = 20 cycles and hence if both loops run for 5 times, then the entry in the column increments as shown.

	Expression α	Expression β	Expression θ	Expression γ
Process A		10		
Process B		20		
.....				
.....				

	Expression α	Expression β	Expression θ	Expression γ
Process A		30		
Process B		40		
.....				
.....				

.....and so on.

Figure 16 Dynamic entry updates in the NSM and LSMs

Only in such a situation can there be preparedness for unpredictable loop iteration counts.

Case 3: A loop in the macro level i.e. containing more than a single process.

Solution: In this case, there are some control nodes inside a loop. Hence the execution time of the loop changes with every iteration.

This is a much more complicated case than the previous options. Here lets consider a situation where there is a loop covering 2 mutually exclusive paths, each path consisting of 2 processes (A,B & C,D) with (3,7 & 15,5) cycle times.

In the schedule table there will be a column to indicate an entry into the loop and 2 columns to indicate the paths inside the loop. Optimality in scheduling inside the loop can be achieved, but in the global scheme of scheduling, the solution is non-optimal. But this cannot be helped because to obtain a globally optimal

solution, all possible paths have to be unrolled and statically scheduled. This results in a table explosion and is not feasible in situations where infinite number of entries in table are not possible. Hence, from a global viewpoint the loop and all its entries are considered as one entity with the most probable number of iterations considered and the most expensive path in each iteration is assumed to be taken. For example in the above case, path C,D is assumed to be taken all the time.

Now, a schedule is prepared for each path and hence entered into the table under 2 columns. When one schedule is being implemented, the entries for both columns in the next loop iteration is predicted by adding the completion time of the current path to both column entries (of course while doing this care should be taken not to overwrite the entries of the current path while they are still being used). Then when the current iteration is completed and a fresh one is started, the path is realized and the appropriate (updated / predicted) table column is chosen to be loaded form the NSM to the LSMs.

References

[Dasu] "A Tool to identify Possibilities of Prallelism, Pipelining and Reconfigurability", EEE 690 report.

[Emilios] "Extracting Sighificant Patterns from Musical Strings: Some interesting Problems", London String Days 2000 workshop. King's College London.

[Anand] "A graduated assignment algorithm for graph matching", IEEE Trans. PAMI.

[UCLA] "Instruction Generation for Hybrind Reconfigurable Systems", Ryan Kastner et.al, ACM Transactions on Design Automation of Embedded Systems, October, 2002.

[UTA] "Knowledge discovery from Structural Data", Diane J. Cook et. al, Journal of Intelligent Information Systems, 1995.

[George] "An efficient Algorithm for Discovering Frequent Subgraphs", George Karypis et. al; Technical Report 02-026, Department of Computer Science/Army HPC Research Center, University of Minnesota, June 27, 2002.

[Chekuri] "An Analysis of Profile Driven Instruction Level Parallel Scheduling with Application to Super Blocks", Chekuri et. al, In Proceedings of the 29th Annual International Symposium on Microarchitecture (MICRO-29), December 1996.

[Mooney] "Path based Edge Activation for Dynamic Run Time Scheduling", V. Mooney, International Symposium on System Synthesis (ISSS'99), pp. 30-36, November 1999.

[Jha] "Wavesched- A Novel scheduling technique for Control Flow Intensive Designs", Ganesh Lakshminarayana, et. al, ICCAD 1997, 244-250.

[Pop] "Scheduling of Conditional Process graphs for Synthesis of Embedded Systems", Pop, et. al, Design Automation and Test in Europe (DATE '98), February 23 - 26, 1998 Paris, France.

[Delft] "Automatic Detection of Recurring Operation Patterns", Arnold, et. al, Seventh International Workshop on Hardware/Software Co-Design (CODES'99), Rome Italy, May 1999.

ASSIGNMENT

We, Aravind R. Dasu, a citizen of India, residing in Tempe, County of Maricopa, State of Arizona, United States of America, ALI AKOGLU, a citizen of Turkey, residing in Tempe, County of Maricopa, State of Arizona, and SETHURAMAN PANCHANATHAN, a citizen of Canada, residing in Gilbert, County of Maricopa, State of Arizona, United States of America, have made an invention(s) as disclosed in the application for United States patent filed in the U.S. Patent and Trademark Office entitled:

ALGORITHM DESIGN FOR ZONE PATTERN MATCHING TO GENERATE CLUSTER MODULES AND CONTROL DATA FLOW BASED TASK SCHEDULING OF THE MODULES

Acting on behalf of Arizona State University, located in Tempe, Arizona, with an office for conducting business at the Office of Technology Collaborations and Licensing, Arizona State University, P.O. Box 873511, Tempe, Arizona 85287-3511, the Arizona Board of Regents, a corporate body organized under Arizona law, wishes to acquire the entire right and title to and interest in the invention and any improvements on the invention and the above-identified patent application and any patent that may be obtained for the invention.

We, ARAVIND R. DASU, ALI AKOGLU and SETHURAMAN PANCHANATHAN, in accordance with the Arizona State University patent policy, for and in consideration of the sum of $1.00 paid to each of us by the Arizona Board of Regents, and for other valuable consideration, the receipt of which is acknowledged, have sold, assigned, transferred, and conveyed and by this document do sell, assign, transfer, and convey, to the Arizona Board of Regents, its successors and its assigns the entire right and title to, and interest in the invention for which we have made the above-described application and all improvements on the invention, the above-identified patent application, any continued prosecution patent application (CPA), division, continuation or continuation-in-part that claims priority from the above-identified provisional application, and all United States and foreign patents that issue on the above-identified invention and improvements, and all United States and foreign patents and patent applications that claim priority from the above-identified patent application including any patent extensions, renewals or reissues, all for the full term or terms for which any of the foregoing may be granted.

We, ARAVIND R. DASU, ALI AKOGLU and SETHURAMAN PANCHANATHAN, authorize and request the Commissioner of Patents and Trademarks to issue any patents of the United States, resulting from any application described above to the Arizona Board of Regents as the assignee of the rights identified above, for the sole use and benefit of the Arizona Board of Regents, its successors and its assigns.

We, ARAVIND R. DASU, ALI AKOGLU and SETHURAMAN PANCHANATHAN, for the consideration stated above, represent to and agree with the Arizona Board of Regents, its successors, and its assigns, that we have the full power to make this assignment, and that the assigned rights are unencumbered by any previously granted right or license. We, our executors, or administrators will do all acts and things and execute and deliver without further compensation any other instruments, further applications, papers, affidavits, powers of attorney, assignments, and other documents that, in the opinion of the counsel for the Arizona Board of Regents or its successors and its assigns are or may be required to more fully secure and vest the assigned right, title, and interest in and to the Arizona Board of Regents, its successors, and its assigns, and we will sign any applications for utility patent, applications for reissue, continuation-in-part applications or applications for extension or renewal that may be desired by the owner of the patent or patents that may be issued for the invention or improvements on the invention.

Date:_____ By:_____
ARAVIND R. DASU

STATE OF _____)
) ss.:
COUNTY OF _____)

BE IT KNOWN, that on this ____ day of _____, 2003, before me personally appeared ARAVIND R. DASU, known to me to be the person mentioned in and who executed the foregoing assignment, and he acknowledged to me that he executed the same as his free act and deed for the use and purposes therein mentioned.

_____
Notary Public

1117979

Date:_____ By: _____
ALI AKOGLU

STATE OF _____)
) ss.:
COUNTY OF _____)

BE IT KNOWN, that on this ____ day of _____, 2003, before me personally appeared ALI AKOGLU, known to me to be the person mentioned in and who executed the foregoing assignment, and he acknowledged to me that he executed the same as his free act and deed for the use and purposes therein mentioned.

_____
Notary Public

Date:_____ By: _____
SETHURAMAN PANCHANATHAN

STATE OF _____)
) ss.:
COUNTY OF _____)

BE IT KNOWN, that on this ____ day of _____, 2003, before me personally appeared SETHURAMAN PANCHANATHAN, known to me to be the person mentioned in and who executed the foregoing assignment, and he acknowledged to me that he executed the same as his free act and deed for the use and purposes therein mentioned.

_____
Notary Public

1117979